r

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/693,630	10/23/2003	Sriram Subramanian	13768.783.230	1530

47973 7590 07/17/2007
WORKMAN NYDEGGER/MICROSOFT
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UT 84111

EXAMINER	
WOODS, ERIC V	
ART UNIT	**PAPER NUMBER**
2628	

MAIL DATE	DELIVERY MODE
07/17/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No. 10/693,630	Applicant(s) SUBRAMANIAN ET AL.	
	Examiner Eric Woods	Art Unit 2628	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *23 April 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-35 and 68* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-35 and 68* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

Continued Examination Under 37 CFR 1.114

A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 4/23/2007 has been entered.

Response to Arguments

Applicant's arguments, see Remarks pages 8-10 and claim amendments, filed 4/23/2007, with respect to the rejection(s) of claim(s) 1-35 under 35 USC 103(a) have been fully considered and are persuasive.

Therefore, in view of **applicant's amendments to the claims**, the rejection of claims 1-35 under 35 USC 103(a) has been withdrawn.

However, upon further consideration, a new ground(s) of rejection is made in view of various references as set forth below.

Non-elected, withdrawn claims 36-67 have been canceled.

Claim 68, a computer-readable medium implementing the method of claim 1, has been added.

*The following rejections are made based on the following logic: a list containing of elements comprising: B ... and A-based (equivalent to A-derived) sets must therefore exclude B from being derived from or based upon A. Therefore, based on the file wrapper – specifically, claim 1 as written as presented before the instant amendment, "markup in native format" is understood to be XML-based. Thusly, the contradiction is created that A and B cannot be related.

Claim Rejections - 35 USC § 112

The following is a quotation of the first paragraph of 35 U.S.C. 112:

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 1-35 and 68 are rejected under 35 U.S.C. 112, first paragraph, because the specification, while being enabling for most claimed limitations (with respect to claims 1 and 68, the only independent claims), does not reasonably provide enablement for "markup language in native format" wherein such markup is not XML-based. The specification does not enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or use the invention commensurate in scope with these claims.

Claims 1-35 and 68 stand rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter that was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention. Specifically, the recitation of "the parser/translator enabled to interpret markup language data in native format ... and XML-based markup" is not supported, where the markup language is NOT XML-based or XML-derived, given that 'based' has a very wide interpretation.

Claims 2-35 are rejected for not correcting the deficiencies of their parent claims.

The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

Claims 1-35 and 68 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claims 1 and 68 are found to be indefinite for the following reason(s):

-The parser/translator is 'enabled to interpret' **markup language data in native format...and XML-based markup**. The problem here is that all versions of markup languages that are known are based upon, derived from, a subset of, or an implementation or extension of XML (eXtensible Markup Language)-the base description for all markup languages -- including SVG, HTML, UML, X3D, VRML, etc. Therefore, given that XML is the basis from which all standardized markup languages have been derived, it is unknown and totally unclear how a markup language in 'native format' could be different than XML-based markup, in the sense that 'based' is construed as above

-The claim impermissibly mixes statutory categories of invention – e.g. a method and a system; see MPEP 2173.05(p)(II), citing *IPXL Holdings v. Amazon.com, Inc*, 430 F.2d 1377, 1384, 77 USPQd 1140, 1145 (Fed. Cir. 2005), in the description of a **parser / translator** and **the media integration layer**: in order to be permissible those particular claim element needs to be rewritten –examiner would suggest, e.g. "interpreting, via a parser / translator, ..." and the media integration layer clause needs to be rewritten in a manner reflecting its purpose (examiner suggests, with respect to clause 1, combining clause 1 and clause 2 in such a way to read "...the media integration layer that is among...")

Claims 2-35 are rejected as failing to correct the deficiencies of their parent claim(s).

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 1 and 68 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Beda et al (US PGPub 2003/0076329 A1) in view of Lewallen (US 6,675,230 B1), which incorporates the SVG specification by reference.

As to claims 1 and 68 (method and computer readable medium implementing method),

Beda teaches:

A computer-implemented method for arranging vector graphics data for processing into an output, the method comprising: (Beda [0072-0073] teaches storing vector graphics in a "primitive list" or primitive container.)

-**Receiving a function call via an application programming interface (API) of a media integration layer (MIL),** (MIL has API - Beda teaches Media Integration Layer in [0039-0041], that has APIs by which higher level programs (e.g. user GUI in OS) access MIL [0040] to render output graphics [0011,0013,0017], where such API has functions [0070,0111,0113] that are called (Abstract, [0014]), and the API receives such calls [0017,0039-0041])

-**The media integration layer being among a plurality of layers in a graphics processing environment, the media integration laver (MIL) comprising a plurality of types of objects, the objects including a VisualManager object which manages the rendering a Visual Tree to a medium,** (Beda MIL has multiple layers [0041], where the 'high-level' and 'low-level engines comprise 'layers' (e.g. layer=level) [0043,0046,**0048**], which generates a plurality of components in tree form ([0044,0057]; Figure 4, element 300; Figure 5, element 500; Figure 2, block 206 generates the tree [0043]. The system has a display manager object 420 (Figure 4) that manages the tree [0066] by utilizing a pointer to the root container of the tree, additionally as in Figure 6, element 604 [0091])

-**The function call comprising graphics-related data;** (Beda [0017])

-**Causing a change in a graphics display in response to the modification of data in the scene graph.** (Beda [0011,0012,0016])

Beda teaches the use of a **scene graph** ([0016]) where the **API function calls** (Abstract, [0014, 0016,0041,0080]) **cause data in a scene graph data structure to be modified** ([0016,0056]), where the **function calls contain graphics-related data** ([0014,0016,0041-0043,etc]) but fails to teach, where Lewallen teaches:

-A parser/translator interpreting the function call comprising graphic-related data, the parser/translator enabled to interpret markup language data in native format, direct code calls, object model code calls, and XML-based markup; based on the function call; and (Lewallen e.g. Figure 1, where the Bridge 4 could be regarded as a layer, and clearly this layer receives calls ('mixed statements') and sorts them, sending direct Java object calls to the JVM 22 and the other code to the UI 10 as necessary and described below, wherein clearly the Figure 1 of Lewallen shows a plurality of layers, and Lewallen (Abstract, 3:25-4:20) discusses that the system is designed to render graphics ('graphics processing environment'). The data structure of Lewallen is in **tree** format, which resembles the recited scene graph – e.g. Lewallen 4:7-20 teaches a **parser / translator** (11:50-63). As noted above, Lewallen teaches SVG, which is a two-dimensional vector **graphic markup language**, and clearly Lewallen is directed to graphical environments (1:5-3:25), where clearly any APIs exposed (including W3C APIs) involve graphical processing environments. Lewallen teaches mixed-statement programs 2a/b/c (which comprise Java programming language) that includes standard API interfaces developed by the W3C for the DOM (e.g. Document Object Model) and language statements from different programming languages and/or protocols (note

5:48-40, where it is stated that mixed statement programs are written to include Java programming language statements as well as W3C API interface calls), such as Java and non-Java standard API interfaces (5:1-10). See below (5:11-25 quoted):

The DOM model is a standard interface used to define the structure of documents, particularly HTML and XML documents. In the DOM specification, the term "document" is used in the broad sense to include the components of a textual document as well as components of an application program. The DOM interface represents the document or application program as a hierarchical arrangement of nodes. All the components of an HTML or XML document, including data as well as program elements, such as the user interface elements, can be expressed as hierarchically arranged nodes. The W3C DOM specifications provide API interfaces to access, change, delete or add nodes to the DOM representation of a document, or application program. The API interfaces specified in the DOM specifications are referred to herein as "W3C API interfaces".

As shown therein, clearly the parser can interpret calls to the DOM API, which clearly represents **object model code**. Further, Java programming statements clearly correspond to "**direct code calls**", where calls to Java objects and Java programming statements are forwarded directly to the Java Virtual Machine (JVM) 22. Note Lewallen 7:1-20, with respect to Java **native** objects and the like. Furthermore, since the markup includes W3C DOM APIs, it must be in **native** format. Finally, **SVG is a form of XML, e.g. XML-based markup**. The XML **must** be interpreted to create the internal representation through the DOM such that it creates the representation where operations upon SVG objects can be made. For example, the alternate embodiment of the bridge 4 in Figure 1 is shown in Figure 4, wherein SVG is taken and transformed to the UI document APIs, such that there is a UI document 226 maintained that is a DOM document tree, wherein it provides the location for the various **object model code calls** – (see 11:50-12:20), where the internal state of the document can be manipulated by a particular SVG section and the like, as explained in the cited paragraph immediately above, where the UI elements can be in Java, where clearly any calls that can 'access, manipulate, create, modify. The entire document is exposed to the application (9:40-10:25, note 10:10-25), wherein this clearly constitutes '**native format instructions**'. Lewallen clearly teaches a **parser/translator** that maps elements in the different levels of markup and mixed programs (2a, 2b, 2c) to various objects and elements within the DOM model, where this clearly is a translation operation (4:7-21, 12:50-54), where clearly the standard APIs allow access to underlying Java objects and the like. Clearly, as noted above, the DOM tree is intermediate (12:18-60), where the elements represent other items at a deeper level in hierarchy. **More specifically, "....the Java developer is allowed to expose data in any object in the user interface, including DOM trees, to java tools...may include Java Database Connectivity (JDBC) to access data from a database..."**(9:10-25).) That is, different levels of the hierarchy can be exposed to upper level ones for operations, as UI objects 14 and elements 10 being mapped to upper level objects via java and the like (7:1-20), thusly showing at least a deeper level, at least conceptually, in Figure 4 (somewhat in Figure 1). Clearly, Lewallen teaches **receiving function calls via W3C APIs** that can manipulate object model level objects, and deeper level Java objects (where the UI objects can be Java).Lewallen clearly teaches that one variant of the Bridge (e.g. element 200 in Figure 4) takes the DOM object calls and sends them to the UI API, which creates **UI document object 226,**

which contains an embedded DOM document tree based on the SVG engine. This constitutes 'an element tree of elements', where these elements have associated property data and correspond to object element models. It is clear that SVG itself represents markup, and commands in SVG are in markup format, but that SVG commands per se can be calls to the object model (e.g. SVG code in W3C API command format), where such are W3C API commands, wherein such API interfaces include numerous methods to implement objects in the UI 10, e.g. "exposing a Java program or mixed statement program (2a, 2b, 2c in Figure 1) to the W3C API interfaces, such mixed statement program containing Java program statements can access any user interface feature and object that the user interface program 10 is capable of implementing. Thus, with the preferred computer architecture, the Java program is no longer constrained to the Java programming space, and may extend the Java program to other objects and programs available in commonly used user interface programs...include...underlying UI objects...". To clarify, see Figure 7, where a call is made via a W3C API in SVG format. Further, the reference clearly says that W3C API calls can generate native format Java objects and the like (7:34-8:55), **see specifically 14:50-67**, quoted in part: "For instance, if the Internet Explorer receives an Internet Explorer createElement command to create an SVG element, then the Internet Explorer stub factory would call the SVG stub factory to create the new SVG element and the Java object for this SVG element, which the Internet Explorer stub factory would then include as a node in the DOM tree. If the new element, e.g. SVG element comprised a DOM tree of elements maintained in a separate file, then the next W3C command in the mixed statement program would likely be a command to set a SRC (source) attribute for the newly created element..." This **clearly** establishes that the intermediate format (e.g. the element tree) is created and modified by the W3C API commands, which are **clearly** in native format as specified above, since the API takes them in as native commands as discussed therein.)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Beda in view of Lewallen because Lewallen teaches that in a mixed-language program that Java commands are passed down to the JVM, but then further teaches that the DOM tree objects and such have mappings to Java objects, which means that they are in the end being passed to underlying Java control layers, where this allows the application developer more manners in which to interact with the scene graph data structure. See Lewallen Abstract, where Lewallen solves the problems with the current art – e.g. network-based graphics- set forth in 2:8-3:10 by allowing objects to have their own standard APIs, as discussed in Lewallen 3:25-52.

WITH RESPECT TO ALL DEPENDENT CLAIMS THAT FOLLOW BELOW, THE INDEPENDENT CLAIM ABOVE CLEARLY TEACHES USING FUNCTION CALLS TO PERFORM STEPS IN A GRAPHICAL ENVIRONMENT. THAT LIMITATION HAS BEEN COVERED ABOVE AND ALL PORTIONS OF THE REJECTION FROM CLAIM 1 DEALING WITH THAT ARE INCORPORATED BY

REFERENCE IN THEIR ENTIRETY IN THE REJECTION OF ANY DEPENDENT CLAIM BELOW. THE LEWALLEN REFERENCE CLEARLY ILLUSTRATES IN FIGURE 1 AND AS SHOWN ABOVE THAT ALL OBJECTS HAVE THEIR OWN APIS, ETC. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data elements *prima facie* and inherently possess paint data as set forth by the SVG specification above.

Claims 2-27 and 30-35 are rejected under 35 USC 103(a) as unpatentable over Beda, Lewallen, and SVG as applied to claim 1 above, and further in view of Steele (US PGPub 2004/0110490 A1). **The motivation and rationale for combination of Steele with the above references is provided in claim 2. That motivation and rationale is herein EXPLICITLY INCORPORATED INTO ALL OTHER CLAIMS BELOW BY REFERENCE.**

As to claim 2,

The method of claim 1 wherein causing data in the scene graph to be modified comprises causing initialization of a new instance of a non-drawing visual class.

References Beda, Lewallen, SVG do not implicitly teach this limitation, but Steele implicitly teaches this limitation, that SVG data is decomposed into scene graphs, a.k.a. trees, (see Figure 7), and again – whenever new visual elements enter the scene, new subgroups are instantiated, which *prima facie* (see SVG specification, section 9) are elements that compose visual objects, which therefore are new instances of a visual class as recited above, since a class as recited by applicant is comparable to the basic 'shapes' in SVG (applicant's specification clearly uses it – 23:1-20 where applicant's invention adopts all classes and shapes from SVG) and thusly meets the recited limitation, wherein these are non-drawing visual subclasses.

Specifically, Lewallen discusses SVG in a very general way, whilst Steele provides a specific working implementation of SVG type graphics, that is, expressly showing how such commands are provided and how such data items can be interacted with.

Steele clearly teaches the use of a SVG DOM as element 305 in Figure 3, where that intermediate format is then transferred to the BF object model 315, where this clearly represents a scene graph (see Figure 6), where vector elements 610 and behavior elements 620 exist, where these form scene graph – see Figure 7, which shows the visual elements 610 as a **visual graph** [0051-0058], see 0051-0053 and 0057-0058 specifically. The other elements are represented as a sequence graph 800. These graphs are matched to each other, as in elements 920 and 930, where the original SVG is shown in Figure 910. Clearly the Sequence Graph portion manipulates the visual object layer, such that manipulating the Visual Graph to change locations does the animation of an object, create new objects, etc. SVG can supply animation commands that discuss where the objects are located and the like with attribute and animate commands. Finally, it is clear that the SVG objects shown in Steele have properties associated with them such as size and/or animation information (as an example, see Figure 910). The DOM trees of Lewallen can consist of SVG.

It is well known in the art and shown by Steele in the referenced paragraphs above that SVG objects can have commands such as animation attached to them. Obviously such SVG intermediate data structures (e.g. SVG DOM tree, or SVG DOM in the Steele drawings) have to be translated to a lower, implementation level for actual drawing on the scene. The visibility and/or presence of such objects in the lower level (e.g. scene graph) are obviously controlled by their properties (Steele code as an example)(where such implementations are in Java [0095, 0152]).

It would therefore have been obvious to one of ordinary skill in the art at the time the invention was made to modify Beda in view of Lewallen with the SVG implementations of Steele because this allows for more efficient implementation for looping behavior and fewer modifications to the scene graph [0061-0068].

As to claim 3,

The method of claim 2 wherein causing data in the scene graph to be modified comprises receiving a function call via an interface corresponding to a transform associated with the visual.

Beda and Lewallen do not expressly teach, but Steele teaches this limitation, as he discloses rotations in [0053] and further states that rotations and other transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088] that any visual element or object can modified. Such modifications *prima facie* must associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera [0053]), as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be scaled. (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.)

As to claim 4,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to initialize a new instance of a drawing visual class.

Beda and Lewallen do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the insertion of unique identifiers into media streams [0106], and further [0088] that any visual element or object can modified. Such modifications and insertions *prima facie* must associate code with a suitable / desired insertion as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be logically inserted.

For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061].

As to claim 5,

The method of claim 4 further comprising, causing drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual.

Beda and Lewallen do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing context per se is incorporated into the data structures of Steele (see Figure 7). It further would have been obvious to modify the system of Lewallen to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]). (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.) Clearly Lewallen teaches that the native operating system commands and APIs are used by the user interface (see Figure 1), where this clearly is done via function calls, since "interfaces" in the context of item 16 ("Native O/S Objects and Interfaces") is shown to mean APIs or objects (which have their own function calls)(see the interior function of user interface 10, Lewallen Figure 1).

As to claim 6,

The method of claim 1 further comprising, receiving brush data in association with the function call, and wherein causing data in the scene graph to be modified comprises invoking a brush function to modify a data structure in the scene

graph data structure such that when a frame is rendered from the scene graph, an area will be filled with visible data corresponding to the brush data.

Beda and Lewallen do not expressly teach, but reference Steele does teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which *prima facie* justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. The SVG standard inherently handles these paint limitations. SVG inherently handles these paint limitations.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this

position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation. As to claim 8,
The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1 and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property. As such, reference Steele intrinsically teaches this limitation.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with an image with further details provided in section 11.7.5 under the 'image-rendering' property.

As to claim 11,

The method of claim 10 further comprising, receiving markup corresponding to an image effect to apply to the image.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. The rendering functionality is inherent in SVG – see section 11.7, 14.4, et cetera.

As to claim 12,

The method of claim 1 further comprising, receiving pen data in association with the function call, and wherein causing data to be modified comprises invoking a pen function that defines an outline of a shape.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The term 'pen data' used by applicant above is comparable or analogous to any set of data defining the outline of a shape, including SVG 'path' data. Section 11.3

of the SVG specification sets forth that a user can fill a path that would correspond to the outline of shape with multiple illustrations provided for this under the 'nonzero' and 'even odd' subheadings – see details on paths -- with further details provided in section 11.3 and 11.4 (the individual strokes that create these effects.

As to claim 13,

The method of claim 1 wherein causing the data in a scene graph at least one of the set containing rectangle, polyline, polygon, path, line and ellipse shapes.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include Shape classes. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup.

As to claim 14,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The SVG specification clearly shows in section 9.1 that rectangles are a basic shape, and further that in 9.2 under Example rect02 that such rectangles can have rounded corners, and code is provided that implements such. Also, the 'Rect' class inherently has geometry-related functions as set forth in the beginning to section 9.2. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. As such, reference Steele shows a rectangle 715 in the scene graph in Figure 7 that intrinsically teaches this limitation. Also, said element can be animated under SVG section 19.2. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 15,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Further, the SVG specification sets forth path data in section 9.1 as existing and how a 'path' can define a shape or similar. Both meanings are covered herein. Steele clearly teaches data modification in [0061] as set forth above.

Also, said element can be animated under SVG section 19.2.

As to claim 16,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a line in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. A line element can be animated under SVG section 19.2, which is obviously geometric. Line elements are set forth in SVG section 9.5, and their geometric functions. Steele clearly teaches data modification in [0061] as set forth above.

The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera].

As to claim 17,

The method of claim 1 wherein the markup is related to hit-testing a visual in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, Steele teaches or implies navigation in [0067, 0085] – that is, navigation using a UI through a two-dimensional view, which is what any display normally shows. Therefore, given that a portable computer could clearly be used, the user would clearly be interacting with the display. As such, hit testing would be required for user interactivity, as could the system of Steele under the same rationale. The SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such information is clearly communicated in markup language – see section 16.2 for event types and elements transmitted in markup, for example. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 18,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to transforming coordinates of a visual in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Clearly, SVG teaches the animation of visual elements, see section 19.2, which *prima facie* involves transforming coordinates of a visual in the scene graph data, and according to Steele [0052-0053] and a tree of elements can also be transformed [Steele 0052]. Steele clearly teaches data modification in [0061] as set forth above. The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera].

As to claim 19,

The method of claim 1 wherein the markup is related to calculating a bounding box of a visual in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Bounding box calculations are taught in section 7.1 and detailed in section 7.11 where they are calculated. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 20,

The method of claim 1 wherein causing data in the scene graph be modified comprises invoking a function via a common interface to a visual object in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, the SVG specification teaches interfaces in section 4.3 – there are common DOM interface as set forth there. If the intended meaning of applicant was that such interfaces were based in hardware or software, fundamentally in reference Steele the user interacts with the browser that would provide a common interface, in that all events generated by such browser would go to an interface – that is, Steele clearly sets forth that his invention has various possible interfaces, depending on the embodiment (e.g. PDA, cell phone, et cetera [0004] and [0007-0008]). Steele clearly teaches data modification in [0061] as set forth above. Lewallen can be regarded as implying a common interface via the use of Java.

As to claim 21,

The method of claim 1 further comprising invoking a visual manager to render a tree of at least one visual object to a rendering target.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each element has its own element information that controls rendering aspects. Steele clearly teaches data modification in [0061] as set forth above. It is *prima facie* obvious that a 'visual manager' of some form must exist in order to handle formatting issues and precedence in rendering, and Steele teaches such a manager in [0075-0076. Clearly the rendering information each visual element [Steele 0056-0061] is sufficient such that it is its own 'rendering target' as set forth above.

As to claim 22,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a container object in the scene graph data structure, the contained object configured to contain at least one visual object.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 7 shows a tree derived from an animation is shown – Figure 6 [see Steele 0050 and 0079]. SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use of 'container elements', which are defined as 'An element that can have graphic elements and other container elements as child elements'. Steele clearly teaches data modification in [0061] as set forth above. Clearly, the container object could be the head object of the tree structure shown in Steele Fig. 7.

As to claim 23,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place image data into the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, visual elements can be covered by or tiled with images as established in SVG section 11.1, where SVG teaches: "...can paint (i.e. fill or stroke) with: ...a pattern (vector or image, possibly tiled) ..." which clearly establishes this, with more detail in section 11.7.5 and 11.12.

As to claim 24,

The method of claim 23 wherein causing data in the scene graph to be modified comprises invoking a function to place an image effect object into the scene graph data structure that is associated with the image data.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask for any visual element, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 25,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place data corresponding to text into the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 10.1 of the SVG specification sets forth the use of a 'text' element, and Steele teaches the inclusion of text element 725 in the data tree shown in Fig. 7. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 26,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to provide a drawing context in response to the function call.

Beda and Lewallen do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061]. SVG is also a subset of XML, and SVG teaches metadata use in section 21.1. Steele clearly teaches data modification in [0061] as set forth above.

It further would have been obvious to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]), and the SVG DOM interfaces in section 4.1-4.4 (SVG specification) clearly provide methods for retrieving drawing information, which would be that context.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing – see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele].

As to claim 30,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place animation data into the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 7 shows that each subgroup is shifted a certain amount with x and y coordinates given. Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 19.1 – 19.5 clearly sets forth how each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Therefore, clearly animation data is put into the tree of Fig. 7 Steele, which is clearly a scene graph by every known definition of the term, and a sample SVG XML program is provided in the second page of Fig. 9.

As to claim 31,

The method of claim 30 further comprising communicating timeline information corresponding to the animation data to a composition engine.

Beda and Lewallen do not expressly teach, but reference Steele clearly establishes in [0051-0054] and Figs. 6 and 9 that animation takes place through the SVG standard. Section 19.2 of SVG sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly states that animation has a document start and document end, and further in the second to last paragraph that the SVG system indicates the timeline position of document fragments being animated. Further, according to SVG 19.2.2 the animation is by document fragment and object path, which clearly are passed to the system is specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly, the system of Steele performs compositing and rendering [0007, 0011-0012]. Finally, reference SVG teaches that it supports compositing (section 14.2.1). The composition engine would be, for example, the composition engine of Steele, the display interface of Lewallen, or the like.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where the system in steps 835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079] which performs interpolation during the animation process as set forth in the SVG standard, and *prima facie* the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set forth in the rejection to claim 31 above.

As to claim 33,

The method of claim 1 wherein receiving a function call via an interface of a media integration layer comprises receiving markup, and wherein causing data in a scene graph to be modified comprises parsing the markup into a all to an interface of an object.

Beda does not expressly teach, but as noted in the rejection to claim 1 above, Lewallen is clearly capable of and does receive XML-based markup, which is parsed for parameter and then translated by the UI (Figure 1) into the appropriate functional calls to an interface of an object (see the rejection to claim 1, which is incorporated by reference in its entirety.

As to claim 34,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele does teach it. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 6.18 clearly sets forth aural style sheets, that are audio data that each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by definition, SVG animations would be video.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified comprises changing a mutable value of an object in the scene graph data structure.

Beda and Lewallen do not expressly teach, but reference Steele does teach it. Steele teaches in [0014] that one embodiment of his invention changes the visual graph in accordance to changes of the sequence graph, where the visual graph is comparable to the "scene graph" of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate traversal of this particular limitation.

Claims 28-29 are rejected under 35 USC 103(a) as unpatentable over Beda, Lewallen, and SVG as applied to claim 1, and further in view of Kim et al (US PGPub 2003/0120823 A1)('Kim').

As to claim 28,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a three-dimensional visual into the scene graph data structure.

Beda, Lewallen, and SVG collectively fail to teach this limitation. The Kim reference clearly teaches this limitation. The Kim reference clearly teaches scene graphs as established in the rejection to claim 1 [0007-0009]. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three-dimensional attributes and properties. This *prima facie* establishes that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D specification in software, and, as such, it is software, which *prima facie* uses function calls.

Kim [0007-0008] clearly teaches the use of a scene graph and that X3D requires the construction of such scene graphs from primitives. Kim further teaches that the user can move through a scene [0020, 0026], which clearly establishes that a user is navigating and the scene is constantly being re-rendered, which *prima facie* requires data in the scene graph to be modified.

Kim extols the benefits of three-dimensional graphics in [0001-0007].

Therefore, based on the above teachings, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Beda, Lewallen, and SVG to have three-dimensional elements.

As to claim 29,

The method of claim 28 wherein causing data in the scene graph to be modified comprises mapping a two-dimensional surface onto the three dimensional visual.

Beda, Lewallen, and SVG fail to expressly teach, but the Kim reference clearly teaches this limitation, and X3D standard is only cited to clarify certain points. The Kim reference clearly teaches scene graphs as established in the rejection to claim 1 [0007-0009]. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three-dimensional attributes and properties. This *prima facie* establishes that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D specification in software, and, as such, it is software, which *prima facie* uses function calls. Secondly, the X3D standard clearly allows for the incorporation of 2D images onto three-dimensional elements, as stated in X3D 18.2.1 and 18.4.1, particularly 18.4.1, which reads clearly that "browsers may support other image formats ... which may be rendered into a 2D image" and clearly those images can be applied to three-dimensional objects such as those described in 18.3.1 and as defined in 18.2.1-18.2.3. Motivation and rationale are taken from the rejection to claim 28 above.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Eric Woods whose telephone number is 571-272-7775. The examiner can normally be reached on M-F 8am - 5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Eric Woods 7/2/2007

Ulka Chauhan
Supervisory Patent Examiner

[illegible]

KEE M. TUNG
SUPERVISORY PATENT EXAMINER